

ALIGN.CPP

11

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```

float a_const, b_const;

// the center of the suspect array should translate to...
float dowsample = 1/2.0 - x_transdowsample, same on y??: //
// note that the origin of the dowsampled arrays actually is
// the center of the original array. dowsample=1/2 in the coordinates of the
// original arrays //
x_trans = (float)dowsample;
y_trans = (float)dowsample;

x[4] = (float)(ffridm_dowsample - 1)/(float)2.0 + x_trans;
y[4] = (float)(ffridm_dowsample - 1)/(float)2.0 + y_trans;
b_const = (float)sin((double)rotation*(PI/180.0)/scale);
a_const = (float)sin((double)rotation*(PI/180.0)/scale);

x[0] = x[4] - (a_const*(float)(ydim-1) + b_const*(float)(ydim-1))/(float)2.0;
y[0] = y[4] - (a_const*(float)(ydim-1) + b_const*(float)(ydim-1))/(float)2.0;
x[1] = x[4] - (a_const*(float)(ydim-1) + b_const*(float)(ydim-1))/(float)2.0;
y[1] = y[4] - (a_const*(float)(ydim-1) + b_const*(float)(ydim-1))/(float)2.0;
x[2] = x[4] - (a_const*(float)(ydim-1) + b_const*(float)(ydim-1))/(float)2.0;
y[2] = y[4] - (a_const*(float)(ydim-1) + b_const*(float)(ydim-1))/(float)2.0;
x[3] = x[4] - (a_const*(float)(ydim-1) + b_const*(float)(ydim-1))/(float)2.0;
y[3] = y[4] - (a_const*(float)(ydim-1) + b_const*(float)(ydim-1))/(float)2.0;

return(1);
}

int find_image(
    unsigned char *out,
    int outdim,
    unsigned char *in,
    int inhdim,
    float *x,
    float *y,
    int *x_channels,
    int *y_channels,
    int option
){
    unsigned char *pout;
    int i, j, xx, yy;
    float x_start, y_start, x_scan, y_scan, x_jump, y_jump;
    unsigned char *phi;
    if(option == 1){ // clear template array
        for(i=0; i<(num_channels*outdim*outdim); i++)*(pout++)=(unsigned char)0;
    }
    yaxis_x = x[2]*x[0]/(float)(ydim-1); // this gives the unit vector in terms of the
    yaxis_y = y[2]*y[0]/(float)(ydim-1);
    yaxis_dist = (float)sqrt((double)(yaxis_x*yaxis_x+yaxis_y*yaxis_y));
    xaxis_x = x[1]*y[0]/(float)(xdim-1);
    xaxis_y = y[1]*y[0]/(float)(xdim-1);
    xaxis_dist = (float)sqrt((double)(xaxis_x*xaxis_x+xaxis_y*yaxis_y));
    // starts is origin dist with axes //
    x_start = x[0] + yaxis_x - y[0] + yaxis_y/yaxis_dist;
    y_start = y[0] + xaxis_x - x[0] + xaxis_y/xaxis_dist;
    jump_x = xaxis_x/xaxis_dist/xaxis_dist;
    jump_y = yaxis_y/yaxis_dist/yaxis_dist;
    pout = out;
    for(i=0; i<(outdim-1); i++){
        for(j=0; j<(outdim-1); j++){
            *(pout++) = (int)current_x;
            yy = (int)current_y;
            fracy = current_y - (float)yy;
        }
    }
}

```


[illegible]

```

/* this is a specialized function simply meant to find out which of 4
degrees of orientations is the true orientation of the submain grid.
This is done by comparing the results of the fast Fourier transform
gives this ambiguity in the first place
// rotate orientation
int rotate_orientation(
    unsigned char *data,
    int xdim,
    int ydim,
    int xdim_size,
    int ydim_size,
    int n, // power of 2 used in inverse fft's
    int original_xdim,
    int original_ydim,
    float scale,
    float *scale)
{
    int mult = 1;
    if (n > 1) {
        mult = (float)1.25; // up n to the next higher power of two
        mult = 2;
    }
    float *buffer = new float[n*(n-1)];
    int n2 = n/2-1;
    rotate_scale_image(
        data,
        xdim,
        ydim,
        bump_size,
        original_xdim,
        original_ydim,
        *scale,
        buffer);
    // fft the thing
    of 2 // the size = (int) (log( (double)(n-1) ) / log( 2.0 ) ); // fftdim should always be power
    assumed fft in place buffer, bits, b, w, x; // ultimately, direct calculation may be faster
    // save the original phase values
    float *real = new float[grid_freq_total];
    float *imag = new float[grid_freq_total];
    for (j=0; j<grid_freq_total; j++) {
        real[j] = buffer[n2 + mult*grid_x[j] + 2*n*mult*grid_y[j]];
        imag[j] = -buffer[n2 + mult*grid_x[j] + 2*n*mult*grid_y[j]];
    }
    // now also through the four possible orientations, finding the best fit
    // the current information of this routine is implicitly tied to
    int high, temp, high = (float)-1e20, grid_real, grid_imag;
    float value(4), x_offset(4), y_offset(4);
    // zero out buffer
    memset(buffer, 0, sizeof(float)*n*(n-1));
    for (j=0; j<grid_freq_total; j++) {
        grid_real = (float)cos( (double)grid_phase[j] );
        grid_imag = (float)sin( (double)grid_phase[j] );
        else if (i==1) {
            temp = (j-grid_freq_total/2)*grid_freq_total;
            if (temp < high) {
                if (temp == grid_freq_total/2) grid_imag = (float)sin( (double)grid_phase[temp] );
                else grid_imag = -(float)sin( (double)grid_phase[temp] );
            }
        }
        else if (i==2) {
            temp = (j-grid_freq_total/2)*grid_freq_total;
            if (temp < high) {
                grid_real = (float)cos( (double)grid_phase[temp] );
                grid_imag = -(float)sin( (double)grid_phase[temp] );
            }
        }
        else {
            temp = (j-grid_freq_total/2)*grid_freq_total;
            grid_real = (float)cos( (double)grid_phase[temp] );
            grid_imag = (float)sin( (double)grid_phase[temp] );
        }
        buffer[n2 + mult*grid_x[j] + 2*n*mult*grid_y[j]] = real[j] + grid_real +
        imag[j]*grid_imag;
        imag[j] = grid_real;
    }
}

```

```

float *mellin_mag = new float[n*(n+2)];
for(i=0;i<n*(n+2)/4;i++){fourier_mag[i]=0;}

int count = 0;
int truncated;
for(i=0;i<n;i++){
    for(j=0;j<n;j++){
        count++; data[i*(n+2)+j]=probable_bump_size; // offset to this block
        if(xblocks==0 || yblocks==0){
            truncated = 1; xlength = xblocks;
            else xlength = n;
            if(yblocks==0) ylength = yblocks;
            else ylength = n;
        }
        else { truncated = 0;
            xlength = n;
            ylength = n;
        }
        add_block_magnitude(
            fourier_mag,
            buffer,
            xlength,
            ylength,
            probable_bump_size,
            xdim, // pixel based jump pointer for moving down rows
            truncated
        );
        if(count == total_blocks){xblocks=yblocks;}/this kicks it out
    }
}

// temporary ship this one back for display file
//mempy(mellin_mag,transform,fourier_mag,sizeof(float)*n*(n+2+1));
//return(1);

// now fourier initialise the magnitude profile
//on_joker_remap_public(fourier_mag,mellin_mag,n);
// use temp128 bmp as input alignment template file
//mempy(temp128bmp,in_mag_transform,mellin_mag,sizeof(float)*n*n);
//return(1);

// fourier transform the dog
//realfft2_in_place(mellin_mag,7.0,wr,wl);

load_grid_family(1); // will immediately return if already done
// temporary display results code: this one has a corresponding return inside
load_magpy(mellin_mag,transform,subliminal_grid,sizeof(float)*32*128);
//return(1);

// now compare the patterns
of(int bits = (int) (log( double)(n+1) ) / log( 2.0 ) ); // fftdim should always be power
of(int number_candidates = 20;
float *rotation_buf = new float[number_candidates];
float *value = new float[number_candidates];

gmf(mellin_mag,mellin_mag,transform,n.bits,number_candidates,rotation_buf, scale_buf,value,0);
//return(1);

// temporary display results; matching return in gmf function
// a first check on deciding whether or not a signature/grid is present is possible
// at this point: the ratio between value and value0 should be above some
// threshold: this is unreliable, then complete the alignment/read process,
// then the control is returned to the next figure,
// this will obviously take a longer time to make a negative decision.
delete [] rotation_buf;
delete [] value;
float detection_value = value[0] / value[19];
// there's our empirical data array, false-positive
// curves, true double-entendre negatives, etc.
// if detection value is above some threshold, we have a winner
// if the suspect image has been expanded, scale will be positive
// if the suspect image has not been expanded, scale will come back negative
double increment_pow(2.0 * 0.025);

```

```

scale_buf[0] = (float)pow(increment, (double)scale_buf[0]);

if(blocks == 0 || blocks == 0){
    truncated = 1;
    if(blocks==0)length = xbufsize;
    if(blocks==0)length = ybufsize;
    if(yblocks==0)length = ybufsize;
    else ylength = n;
}

// resolve 90 degree ambiguity in rotation/orientation
// resolve orientation data length, ylength, xdim, probabale_bump_size,
n_xdim, rotation_buf[0], scale_buf[0];
*rotation = rotation_buf[0];
*scale = scale_buf[0];
*present = 1;

//now find precise global alignment parameters
} else { // send back no go on first detect, then get options for quitting or looking harder
    *present = 0;
}

delete [] rotation_buf;
delete [] scale_buf;
delete [] value;
}

return(1);

int experiment(
    unsigned char *data,
    int n
){
    float *img = new float[n*n];

    //for(i=0;i<n;i++)img[i]=((float)0.0;

    load_grid_famly(0); // will immediately return if already done
    realfired_in_place(subluninal_grid,7,0,wr,wi);

    ffd2(subluninal_grid,img,7,0,wr,wi);

    return(1);
}

//main registration program to be used as main module inside other programs */
int main(int argc, char **argv)
{
    unsigned char *template;
    int template_xdim;
    int template_ydim;
    unsigned char *suspect;
    int suspect_xdim;
    int suspect_ydim;
    int num_channels
    if(1){
        //experiment(ttemplate,template_xdim);
        //return(1);

        int present;
        int template_xdim,scale;
        extern float *mellin_mag_transform;
        hunt_for_grid(
            suspect_xdim,
            suspect_ydim,
            1, //channels,
            1, //orient,
            scale,
            1, //downsample
            mellin_mag_transform
        );

        //temporary: place mellin_mag_transform into template for return
    }
}

```

```

// use stemp_bmp as input alignment template file
// load highest-(float)-120,lowest-(float)120;
int i,n=128;
for(i=0;i<(n/2+1);i++){
    if(i%128 < 0) { add((i%128)-(46 < 6) );
    else {
        if (mellin_mag_transform[i].highest>highest,mellin_mag_transform[i],
        i) mellin_mag_transform[i].lowest<lowest,mellin_mag_transform[i],
        i);
        highest = (float)255.0/(highest-lowest);
        for(i=0;i<(n/2+1);i++){
            ((i%128)-(46 < 6) < 6) ? template[i] = (unsigned char)(100,
            i) : template[i] = (unsigned char)( mellin_mag_transform[i] - lowest *highest);
        }
    }
}

// use atemp128.bmp as input alignment template file
float highest=(float)-120,lowest=(float)120;
for(i=0;i<(n+1);i++){
    if (mellin_mag_transform[i].highest>highest,mellin_mag_transform[i],
    i) mellin_mag_transform[i].lowest<lowest,mellin_mag_transform[i],
    i);
    highest = (float)255.0/(highest-lowest);
    for(i=0;i<(n+1);i++){
        template[i] = (unsigned char)( mellin_mag_transform[i] - lowest *highest);
    }
}

else {

    int i,ffridin,bits,array_size,ip,array_size;
    int alignment_mode=2,downsample;
    int rotation=MAX_CANDIDATES; // number of peaks looked at */
    float rotation(MAX_CANDIDATES), scale(MAX_CANDIDATES), value(MAX_CANDIDATES);
    float x_trans(MAX_CANDIDATES), y_trans(MAX_CANDIDATES) * (5), y(5);
    int template_xdim, template_ydim;
    unsigned char *template_lum = new unsigned char(template_xdim*template_ydim);

    // if color image, then create collapse template into a single image.
    // while the real suspect is used during final resampling
    if(template_xdim > 3){
        template = template_lum;
        template_xdim = template_ydim;
        template_ydim = 3;
        for(i=0;i<(template_xdim*template_ydim);i++){
            *(template++) = *pin; // no need for extreme accuracy
            pin++;
        }
        template = suspect_lum;
        for(i=0;i<(suspect_xdim*suspect_ydim);i++){
            *(template++) = *pin; // no need for extreme accuracy
            pin++;
        }
    }

    // find working array size after downsampling (if downsampling is called at all)
    // find the working array size, template_xdim, template_ydim,
    array_size = ffdim(ffridin+2); // the extra 2 is due to the fft routine
    array_size = ip_sampling(ip_sampling+2);
    // using
    // size = (int)(log( double)(ffridin+1) / log( 2.0 ) ); // ffdim should always be power
    // of 2
    // create the template arrays
    float template_real_lum [array_size];
    float template_ip_real = new float[ip_array_size];
    float template_xdim_real = new float[xdim_array_size];
    float suspect_lum = new float[lum_array_size];
    float *stemp = new float[ip_array_size];
    float *suspect_copy = new float[array_size];

    // copy the two inputs into the arrays, with any downsampling and windowing applied
    if(template_xdim > 3){
        ffdin_downsample(
            copy_downsample_window(suspect,suspect_ydim,suspect_xdim,template_xdim,template_real,
            copy_ffridin_downsample);
            copy_downsample(
            copy_downsample_window(suspect,suspect_ydim,suspect_xdim,template_xdim,template_real,
            copy_downsample_window(template_lum,template_xdim,template_ydim,template_real,
            copy_downsample);
            else if(num_channels == 3){
                copy_downsample_window(suspect_lum,suspect_xdim,suspect_ydim,suspect_real,
                copy_downsample_window(template_lum,template_xdim,template_ydim,template_real,
                copy_downsample);
            }
        }
    }
}

```

```

fftDim Downsamps;
memory suspect_Cpy, suspect_real, array_size*sizeof(float) );

// real-valued 2D FFT both suspect and template into it's half-plane complex self
// realifft in place(template_real_bits,0,wr,w1);
// realifft_in_place(suspect_real_bits,0,wr,w1);

// calculate Fourier mellin transform
fourier_mellin_transform(template_real,temp_fftDim,template_ip_real);
fourier_mellin_transform(suspect_real,temp_fftDim,suspect_ip_real);

// realifft in place both suspect and template into 2D FFT each
// realifft_in_place(template_real_bits,0,wr,w1);
// realifft_in_place(suspect_ip_real_ip_bits,0,wr,w1);

// perform generalized matched filter on the two resulting arrays, outputting some number of
// likely candidates, with their associated parameters
// get_matched_candidates(template_real_ip_bits,suspect_ip_real_ip_bits,number_candidates,
//                       guess, scale, value, 0);

// change units on rotation and scale for later stages
for(i=0;i<number_candidates;i++){
    section1(i).fftDim=180.0; // (float)IP.sampling // converts to degrees
    scale1(i) = 1/tempDim*(float)suspect_section1(i).fftDim; // converts to linear scale
}

// now we have a series of candidates ( or 1, and we just need to get the rotation
// and translation information ) wherein one of them should be
// the best match. We will use the rotation and translation information, including both
// the nominal rotation state and the state 180 degrees rotated from the nominal, and
// finds which rotation, scale, and translation gives the highest matched filter
// returns best candidate in first element of rotation, scale, x_trans, y_trans
get_matched_candidates(template_real_ip_bits,suspect_ip_real_ip_bits,
                      suspect_ydim,downsample_rotation,scale,x_trans,y_trans,template_real,
                      suspect_ydim,downsample_rotation,scale,x_trans,y_trans,template_real);

// convert the scale/rotation/translation parameters of the downsampled arrays
// into the x and y positions of the four corners of the suspect array, as projected
// onto the template. The four corners are the four corners of the template, and the
// translation into final alignments to well better than a single pixel, especially
// in light of the subtleties involved with downsampling. The four corners are
// the four corners of the template, and the translation into final alignments to well
// of the suspect, element 1 is the upper right, element 2 lower left, element 3 lower corner
// of the template, and element 4 is the upper left. The template itself, while
// the contribution of the corners to the alignment process is small, the
// point in the x and y arrays is the centroid, used just so you don't have to
// convert to center(x,y,rotation(0),scale(0),x_trans(0),y_trans(0)).
suspect_xdim,suspect_ydim,fftDim,downsample_rotation);

// now fine tune the result using tricky tricks. see notebook of Nov 28, 1995 */
if (num_channels == 1){
    for (int i=0; i<1; i++){
        fine_tune_xdim(template_xdim,template_ydim,suspect_xdim,
                      suspect_ydim,x,y,rotation);
    }
}
else if (num_channels == 3){
    fine_tune_xym(template_xdim,template_ydim,suspect_xdim,suspect_ydim,
                  suspect_ydim,x,y,rotation);
}

// that but best, create the output image array, with various options */
// first, suspect_ydim,x,y,num_channels(1); // 1, stands for aligned suspect with black everywhere else

/* Record some results of the alignment process in our status structure */
m_alignmentStatus.rotation = rotation(0);
m_alignmentStatus.y_scale = scale(0);
m_alignmentStatus.x_scale = scale(0);
m_alignmentStatus.x_trans = x_trans(0);
m_alignmentStatus.y_trans = y_trans(0);

delete m_all;
delete m_template_real;
delete m_template_ip_real;
delete m_suspect_Cpy;
delete m_suspect_ip_real;
delete m_fftDim;
delete m_suspect_xdim;
delete m_suspect_ydim;
delete m_template_xdim;
delete m_template_ydim;

return(1);

```

```

/* shall do at least get the main registration program up and running, tested
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
main()
{
    // new Geoff's testing purposes this main() function was used to
    // create a stand-alone program which exercised the alignment
    // algorithm. This is lifted out for the windows version.
    main_int argc, char *argv[];

    {
        int template_xdim,template_ydim,suspect_xdim,suspect_ydim,
            template_filename(80),suspect_filename(80);

        printf("Template file name please: ");
        scanf("%s",template_filename);
        printf("X dimension and Y dimension of template file: ");
        scanf("%d %d",&template_xdim,&template_ydim);
        printf("X dimension and Y dimension of suspect file: ");
        scanf("%d %d",&suspect_xdim,&suspect_ydim);

        unsigned char *img = new unsigned char[template_xdim*template_ydim*sizeof(tum)];
        unsigned char *imgt = new unsigned char[suspect_xdim*suspect_ydim*sizeof(tum)];

        // read in binary data into complete */
        if(read_img(template_filename,"img"))
            if(!img) return -1;
        if(!imgt) return -1;

        // read in "center_register.can't open template_filename",
        exit(1);

        //read_img_alsoef(unsigned char*,template_xdim*template_ydim,intf);
        fclose(img);

        inf = fopen(suspect_filename,"rb");
        if(!inf)
        {
            fprintf(stderr,"register: can't open %s\n",suspect_filename);
            }
        //reading sizeof(unsigned char),suspect_xdim*suspect_ydim,intf);
        fclose(inft);

        /* saving registered image inside array 'template' */
        direct_registration(img,template_xdim,template_ydim,imgt,suspect_xdim,suspect_ydim);

        //write out binary data from template */
        inf = fopen("reg_out","wb");
        if(!inf){fprintf(stderr,"register: can't open %s\n",reg_out");
                exit(1);
            }
        fwrite(img,sizeof(int),template_xdim*template_ydim,intf);
        fclose(inft);

        /* free and clean up */
        delete img;
        delete inft;

        return(0);
    }
}

#endif //NEED_MAIN

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////
FILES: Align.h
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////
DESCRIPTION:
Header file for the Alignment core algorithm code and the "Align"
C++ class used to encapsulate this code.

The Alignment code is equivalent to Geoff Rhoads' "register" core
program on the SGI, then ported to Minisv and Visual C++ as a "console" program,
and finally incorporated into the Signer window application.

Copyright (c) 1996 Dynamac Incorporated. All rights reserved.
```

[illegible]

[illegible]

```

// .....
lpPal->paiEntry[i].peRed = lpPal->palColors[i].rRed;
lpPal->paiEntry[i].peGreen = lpPal->palColors[i].rGreen;
lpPal->paiEntry[i].peBlue = lpPal->palColors[i].rBlue;
lpPal->paiEntry[i].peAlpha = 0;
}
else
{
    lpPal->paiEntry[i].peRed = lpPal->palColors[i].rRed;
    lpPal->paiEntry[i].peGreen = lpPal->palColors[i].rGreen;
    lpPal->paiEntry[i].peBlue = lpPal->palColors[i].rBlue;
    lpPal->paiEntry[i].peAlpha = 0;
}
}

// create the palette and get handle to it
hResult = Pal->CreatePalette(lpPal);
if (GlobalLock((HGLOBAL) hResult))
{
    GlobalUnlock((HGLOBAL) hResult);
}

return hResult;
}

// .....
// FindBits()
// Parameter:
// lpBmp - pointer to packed-DIB memory block
// Return Value:
// lpBmp - pointer to the DIB bits
// Description:
// This function calculates the address of the DIB's bits and returns a
// pointer to the DIB bits.
// .....
LPSTR WINAPI FindBits(LPSTR lpBmp)
{
    return (lpBmp + *LPDWORD)lpBmp + ::PaletteSize(lpBmp);
}

// .....
// DIMid()
// Parameter:
// lpBmp - pointer to packed-DIB memory block
// Return Value:
// DWORD - width of the DIB
// Description:
// This function gets the width of the DIB from the BITMAPINFOHEADER
// structure. If the DIB is from the BITMAPCOREHEADER
// structure, it is an other-style DIB.
// .....
DWORD WINAPI DIMid(LPSTR lpBmp)
{
    LPBITMAPINFOHEADER lpBmp; // pointer to a Win 3.0-style DIB
    LPBITMAPCOREHEADER lpBmp; // pointer to an other-style DIB

    /* point to the header (whether old or Win 3.0) */
    lpBmp = (LPBITMAPINFOHEADER)lpBmp;
    lpBmp = (LPBITMAPCOREHEADER)lpBmp;

    /* return the DIB height if it is a Win 3.0 DIB */
    if (IS_WIN30_DIB(lpBmp))
        return lpBmp->biHeight;
    else
        return lpBmp->bcHeight;
}

// .....
// PaletteSize()
// Parameter:
// lpBmp - pointer to packed-DIB memory block
// Return Value:
// WORD - size of the color palette of the DIB
// Description:
// This function gets the size required to store the DIB's palette by
// multiplying the number of colors by the size of an RGBQUAD (for a
// Win 3.0-style DIB) or by the size of an RGBTRIPLE (for an other-
// style DIB).
// .....
WORD WINAPI PaletteSize(LPSTR lpBmp)
{
    /* calculate the size required by the palette */
    if (IS_WIN30_DIB(lpBmp))
        return (DWORD)((::DIBNumColors(lpBmp) * sizeof(RGBQUAD)));
    else
        return (WORD)((::DIBNumColors(lpBmp) * sizeof(RGBTRIPLE)));
}

// .....
// DIBNumColors()
// Parameter:
// lpBmp - pointer to packed-DIB memory block
// Return Value:
// WORD - number of colors in the color table
// Description:
// This function calculates the number of colors in the DIB's color table
// by finding the bits per pixel for the DIB (whether Win 3.0 or other-style
// DIB). If bits per pixel is 1: colors=2; if 4: colors=16; if 8: colors=256;
// if 24: no colors in color table.
// .....

```

```

.....//
WORD WMINF1 DIBNUMCOLORS(LPSTR lpb1)
WORD wBitCount; // DIB bit count

/* If this is a Windows-style DIB, the number of colors in the
 * color table can be based on the number of bits per pixel
 * color table. If this is the case, return the value.
 * If this is the case, return the appropriate value.
 */
if ((IS_WIN30_DIB(lpb1))
    DIBSD dciUsed;
    declUsed = ((LPBITMAPINFOHEADER)lpb1)->biClrUsed;
    if (declUsed != 0)
        return (WORD)declUsed;
    else
        return (WORD)wBitCount;

/* Calculate the number of colors in the color table based on
 * the number of bits per pixel for the DIB.
 */
if ((IS_WIN30_DIB(lpb1))
    else wBitCount = ((LPBITMAPINFOHEADER)lpb1)->biBitCount;
    else wBitCount = ((LPBITMAPCOREHEADER)lpb1)->bcBitCount;

/* Approximate number of colors based on bits per pixel */
switch (wBitCount)
{
    case 1:
        return 2;
    case 4:
        return 16;
    case 8:
        return 256;
    default:
        return 0;
}
}

.....//
Parameter:
LPSTR lpb1 - pointer to packed-DIB memory block
Return Value:
WORD - number of bits per pixel
Description:
Added by Clay Davidson 11/7/95. Simply returns the number of bits per
pixel (i.e., 2, 4, 8, 24), regardless of the state of the color table.
WORD WMINF1 DIBNUMCOLORS(LPSTR lpb1)
WORD wBitCount;
WORD wBitCount;
if ((IS_WIN30_DIB(lpb1))
    else wBitCount = ((LPBITMAPINFOHEADER)lpb1)->biBitCount;
    else wBitCount = ((LPBITMAPCOREHEADER)lpb1)->bcBitCount;
    return wBitCount;
}

.....//
// Clipboard support
.....//
Function: CopyHandle (from EX Diaview sample clipboard.c)
Purpose: Makes a copy of the given global memory block. Returns
a handle to the new memory block (NULL on error).
Routine stolen verbatim out of ShowDIB.

```

```

// Params: h == Handle to global memory to duplicate.
// Returns: Handle to new global memory block.
//-----//
HANDLE WINAPI CopyHandle (HANDLE h)
{
    BYTE *lpCopy;
    HANDLE hCopy;
    DWORD declen;
    if (h == NULL)
        return NULL;
    declen = ::GlobalSize((GLOBAL h));
    if (hCopy = HANDLE) : GlobalAlloc (GMEM, declen) != NULL)
    {
        lpCopy = (BYTE *) : GlobalLock((GLOBAL h) hCopy);
        lp = (BYTE *) : GlobalLock((GLOBAL h) h);
        while (declen-->0)
            *lpCopy++ = *lp++;
        : GlobalUnlock((GLOBAL h) hCopy);
        : GlobalUnlock((GLOBAL h) h);
    }
    return hCopy;
}

// dibapi.h
// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992 Microsoft Corporation
// All rights reserved.
// This source code is only intended as a supplement to the
// Windows SDK and/or Visual C++ Developer's documentation.
// Guidelines and/or Winhelp documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

#ifdef _INC_DIBAPI
#define _INC_DIBAPI
// Handle to a DIB */
DECLSPEC_HANDLE(HDIB);
/* DIB constants */
#define PALVERSION 0x300
/* DIB Macros */
#define IS_WIN30_DIB(lpb1) ((LPDWORD)(lpb1) == sizeof(BITMAPINFOHEADER))
#define IS_BITMAP(lpb1) ((lpsect->right - (lpsect->left)
#define IS_BITMAP(lpb1) ((lpsect->bottom - (lpsect->top)
#define IS_BITMAP(lpb1) ((lpsect->bottom - (lpsect->top)
/* WIDTHBYTES performs DWORD-aligning of DIB scanlines. The "bits"
 * parameter is the number of bits per scanline. The "padding"
 * parameter is the number of bytes to pad the scanline to, and
 * this macro returns the number of DWORD-aligned bytes needed
 * to hold those bits.
 */
#define WIDTHBYTES(bits) (((bits) + 31) / 32 * 4)
/* Function prototypes */
BOOL WINAPI FindDIB (HDC, LPRECT, HDIB, LPRECT, COLORREF *pPal);
BOOL WINAPI FindDIB (HDC, LPRECT, HDIB, LPRECT, COLORREF *pPal);
LPSTR WINAPI FindDIBData (LPSTR lpb1);
DWORD WINAPI DIBWidth (LPSTR lpb1);
DWORD WINAPI DIBHeight (LPSTR lpb1);
WORD WINAPI PaletteSize (LPSTR lpb1);
WORD WINAPI DIBNumColors (LPSTR lpb1);
WORD WINAPI DIBNumColors (LPSTR lpb1);
HANDLE WINAPI CopyHandle (HANDLE h);
HANDLE WINAPI ReadDIB (HDIB, LPSTR, LPSTR, LPSTR);
HDIB WINAPI ReadDIBFile (LPSTR lpFile);
#endif // _INC_DIBAPI

```


[illegible]

```

        }
        j1 = (j-cbints)*1 ;
        xi = a[i1] ;
        ar[i1] = ar[j1] ;
        ar[j1] = xi ;
        ar[i1] = xi ;
    }

    ffc( sar[0], sar[0], nbits, inv, wr, w1, 1 )

    for( i = 1 ; i <= n ; i++ )
    {
        for( j = 0 ; j < i ; j++ )
        {
            j1 = (j-cbints)*1 ;
            xi = a[i1] ;
            ar[i1] = ar[j1] ;
            ar[j1] = xi ;
            ar[i1] = xi ;
        }
    }

    ffc( sar[0], sar[0], nbits, inv, wr, w1, 0 ) ;

    return(0) ;

void reallf_two_arrays(float *arraya, float *array2, int nbits, int inv, float *w, float *w1, int neww)
{
    register int j ;
    register int i ;
    register int nhalf ;
    register float *p ;
    register float *pcomp ;
    register float *pcomp2 ;
    register float *p1 ;
    register float *p2 ;
    register float float_p1 ;
    register float float_p2 ;
    register float float_pcomp1 ;
    register float float_pcomp2 ;

    n = 1 << nbits ;
    nhalf = n/2 ;

    if(!neww){
        float *arraya2[nbits, inv, wr, w1, neww] ;
        /* sort the results */
        pcomp = temp1 ;
        pcomp2 = temp2 ;
        p1 = array1 ;
        p2 = array2 ;
        pcomp1 = *p1++ ;
        pcomp2 = *p2++ ;
        p1 = array2[n-1] ;
        p2 = array1[n-1] ;
        pcomp1 = *p1-- ;
        pcomp2 = *p2-- ;
        for(i=1; i<nhalf; i++){
            *pcomp1++ = float0.5 * (*p1 + *p2) ;
            *pcomp2++ = float0.5 * (*p2 - *p1) ;
            *p1 = array2[n-half-i] ;
            *p2 = array1[n-half-i] ;
        }
        /* now copy the results back into original arrays */
        memcpy(array, temp1, sizeof(float)) ;
        memcpy(array2, temp2, sizeof(float)) ;
    }
    else /* re-sort results */
    {
        pcomp = temp1 ;
        pcomp2 = temp2 ;
        p1 = array1 ;
        p2 = array2 ;
    }
}

```

[illegible]

[illegible]

[illegible]

```

// tag := Image (IDB1 IDB3)
BitmapInfo *bi;
m_hBitmapData = NULL;
m_bFileOk = TRUE; // its already been opened
m_IDB1 = IDB1;
m_IDB3 = IDB3;
m_nIDB1 = (LSTN) : GlobalLock (GLOBAL1_m_IDB1);
// NOTE: THE FOLLOWING MEMBER POINTERS ARE ONLY VALID WHILE
// INFORMATION IS BEING READ. IF YOU WANT TO WRITE INFORMATION,
// I LEAVE THE DATA LOCKED UNTIL THE OBJECT IS DESTROYED.
bi = info = (BITMAPINFO *) m_IDB1;
// Set up a pointer to the BITMAPHEADER and REQUEST array.
m_pBitmapHeader = (BITMAPHEADER *) bi->bmiHeader;
m_pRequestArray = (REQUEST *) bi->bmiColors + 4 * m_Info->biColors[0];
// Get the pointer to the image data.
m_pData = (BYTE *) bi->bmiColors[0];
m_pDataBits = (unsigned char *) : FindDIBits(m_pIDB1);
m_pBmpHeader = m_pBitmapHeader->bmiHeader;
m_nXMin = m_pBmpHeader->biWidth;
m_nYMin = m_pBmpHeader->biHeight;
m_nCompression = m_pBmpHeader->biCompression;
m_nWidthInBytes = WIDTHBITS * m_nXMin * m_pBmpHeader->biBitCount;
}
// tag := Image (IDB1 IDB3)
// Constructors which creates an Image object, given the name of a DIB
// or BMP file.
// tag := Image (Existing filename)
CFile *pFile;
CFileException *pFileException;
m_nFileOk = TRUE;
m_hBitmapData = NULL;
n_bOpenData = FALSE;
if (!file.Open(fileName, CFile::modeRead | CFile::shareDenyWrite, false,
    "r"))
{
    CFileException *pError = new CFileException("Error reading Image file.");
    m_nFileOk = FALSE;
    m_nCompression = FALSE;
    m_nFileOk = TRUE;
}
else
{
    m_nFileOk = TRUE;
}
// Try to read the DIB file, catch any exceptions.
TRY
{
    m_IDB1 = : ReadDIBFile(file);
    ON_THROW(CFileException, &word)
    file.Abort();
    m_nFileOk = NULL; // error reading the image file", NULL,
    m_nCompression = FALSE;
    m_nFileOk = TRUE;
    m_nCompression = FALSE;
}
CATCH(CFileException, &word)
END_CATCH
m_IDB1 = (LSTN) : GlobalLock (GLOBAL1_m_IDB1);
// NOTE: THE FOLLOWING MEMBER POINTERS ARE ONLY VALID WHILE
// INFORMATION IS BEING READ. IF YOU WANT TO WRITE INFORMATION,
// I LEAVE THE DATA LOCKED UNTIL THE OBJECT IS DESTROYED.
bi = info = (BITMAPINFO *) m_IDB1;
// Set up a pointer to the BITMAPHEADER and REQUEST array.
m_pBitmapHeader = (BITMAPHEADER *) bi->bmiHeader;
m_pRequestArray = (REQUEST *) bi->bmiColors + 4 * m_Info->biColors[0];
// Get the pointer to the image data.
m_pData = (BYTE *) bi->bmiColors[0];
m_pDataBits = (unsigned char *) : FindDIBits(m_pIDB1);
m_nXMin = m_pBmpHeader->biWidth;
m_nYMin = m_pBmpHeader->biHeight;
m_nCompression = m_pBmpHeader->biCompression;
m_nWidthInBytes = WIDTHBITS * m_nXMin * m_pBmpHeader->biBitCount;
}

```

[illegible]


```
// Need public access to the CMOFrameWnd; OnCreateUpCreateStruct() function.  
void myCmWindowWnd(void) {OnWindowWnd()};  
  
protected // control bar embedded members  
CMOFrameWnd m_CmWindow;  
CToolbar m_mToolbar;  
  
// Generated message map functions  
protected:  
//AFX_MSG_MAP(CMOWindow)  
afx_msg void OnPaint();  
afx_msg void OnIdle();  
afx_msg void OnKeyUp(WPARAM wParam);  
//AFX_MSG_MAP  
DECLARE_MESSAGE_MAP()  
};  
  
////////////////////////////////////  
  
// myCmWindow.cpp : implementation file  
// This class was created in order to over-ride the  
// default behavior of the CMOWindow::PreCreateWindow()  
// method, so that we can add our own class to create  
// a customized child window title.  
  
#include "stdafx.h"  
#include "myCmWindow.h"  
  
#ifdef _DEBUG  
static char BASED_CODE THIS_FILE[] = __FILE__;  
#endif  
  
// CMOWindow  
IMPLEMENT_DYNAMIC(CMOWindow, CWinThread)  
  
CMOWindow::CMOWindow()  
{  
    CMOWindow::~CMOWindow()  
}  
  
BEGIN_MESSAGE_MAP(CMOWindow, CWinThread)  
    //AFX_MSG_MAP(CMOWindow)  
END_MESSAGE_MAP()  
  
BOOL CMOWindow::PreCreateWindow(CREATESTRUCT &cs)  
{  
    // Do default processing  
    if (CMOWindow::PreCreateWindow(cs) == 0)  
        return FALSE;  
    else  
    {  
        cs.style |= (LONG) FWS_AUTOTITLE;  
        return TRUE;  
    }  
}  
  
////////////////////////////////////  
// CMOWindow message handlers  
////////////////////////////////////  
  
// myCmWindow.h : header file  
//  
#ifndef MYCWINDOW_H  
#define MYCWINDOW_H  
  
class CMOWindow : public CWinThread  
{  
public:  
    CMOWindow();  
    ~CMOWindow();  
};  
#endif
```



```
// PacketMsg = PackCheck(msg_length)
// int i;
// }
// }
// BitsOfInt()
// Function which reads the recovered bit array, containing one bit of
// the packed binary message in each char element, and packs these bits
// back into a single integer array. It then converts the compacted msg
// character per element into a long integer array.
// ASCII and puts the resulting characters in the m_recoveredAsciiMsg
// array.
// This is recovered and stored in the m_recoveredChecksum variable
// and packing: (bitsOfInt(msg))
{
    unsigned char *p_read_bits, *p_signed_bits;
    unsigned char bit;
    // First, build the m_compacting array from the m_readersBinaryArray.
    p_read_bits = m_readersBinaryArray;
    p_signed_bits = m_unsignedArray;
    for (int i = 0; i < m_msglength; i++)
    {
        m_compactMsg[i] = 0; // Start with nothing.
        for (j = PACKED_BITS_PER_CHAR - 1; j >= 0; j--)
        {
            if (*p_read_bits == 1)
                m_compactMsg[i] |= (bit << j);
            else
                m_compactMsg[i] |= (bit << j);
            // Compute bit success rate metric:
            if (*p_read_bits == m_recoveredBits)
                m_correctBits++;
            p_read_bits++;
            p_signed_bits++;
        }
    }
}

// Now recover the checksum from the end of the bit array.
m_recoveredChecksum = 0;
for (j = PACKED_BITS_PER_CHAR - 1; j >= 0; j--)
{
    if (*p_read_bits == 1)
        m_recoveredChecksum |= (1 << j);
}
// Compute bit success rate metric:
if (*p_read_bits == *p_signed_bits)
    m_correctBits++;
p_read_bits++;
p_signed_bits++;
}

// Next, convert the compact form to an ASCII string.
for (int i = 0; i < m_msglength; i++)
{
    if (m_compactMsg[i] >= zero && m_compactMsg[i] < nine)
        m_recoveredAsciiMsg[i] = 0 + m_compactMsg[i];
    else if (m_compactMsg[i] >= 10 && m_compactMsg[i] < 20)
        m_recoveredAsciiMsg[i] = 1 + m_compactMsg[i];
    else switch (m_compactMsg[i])
    {
        case space:
            m_recoveredAsciiMsg[i] = ' ';
            break;
        case period:
            m_recoveredAsciiMsg[i] = '.';
            break;
        case comma:
            m_recoveredAsciiMsg[i] = ',';
            break;
        case colon:
            m_recoveredAsciiMsg[i] = ':';
            break;
        case slash:
            m_recoveredAsciiMsg[i] = '/';
            break;
        case backslash:
            m_recoveredAsciiMsg[i] = '\\';
            break;
        default:
            // Warn user that an undefined character was found.
            CString warn_msg;
            warn_msg += "Warning: Not part of the DigiMacr character set.";
            MessageBox(NULL, warn_msg, "Warning: Not part of the DigiMacr character set.", MB_OK);
            break;
    }
}
```

```

break;
case m_recoveredasciimsg[i] = '/':
case backslash:
m_recoveredasciimsg[i] = '\\';
break;
default:
// If we don't recognize the character.
break;
}
// Add a null terminator
m_recoveredasciimsg[m_length] = '\0';
// Compute the checksum of the read message
m_computedreaderchecksum = ComputeChecksum(m_compactmsg, m_msglength);
}

// Public member functions
public:
// Constructor: takes user's input message and creates the packed version.
PackedMsg(const char *user_msg);
PackedMsg(int msg_length);

// An accessor allows callers read-only access to the packed msg.
const unsigned char *getPackedMsg(void) const;
int getCompactMsgSize(void) const;
int getMsgLength(void) const;
char *getAsciiMsg(void) const;
char *getReaderAsciiMsg(void) const;
char *getReaderAsciiMsg(void) const;
char *getReaderAsciiMsg(void) const;
int getNumCorrectBits(void) const;
float GetPercentCorrect(void) const;

// Checksum accessors.
unsigned char GetChecksum(void) const;
unsigned char GetReaderChecksum(void) const;
unsigned char GetComputedReaderChecksum(void) const;
int GetMsgLength(void) const;
void BitToBString(void);
// Function to unpack a message, for use by the recognizer ...
void BitFromBString(void);
// Destructor
~PackedMsg(void);

// Private member functions
private:
void PackMessage(void);
unsigned char ComputeChecksum(char *msg, int length);
// Private data
int
Compact_Msg
char
int
unsigned char
char
unsigned char
unsigned char
int
};

#endif // PACKMSG_H
.....

```

```

// FILE: Params.cpp
// DESCRIPTION:
// Implementation of the Parameters classes: SignerParams and ReaderParams.
//
// CREATION DATE: September 8, 1995
// Copyright (c) 1995 Digimarc Incorporated. All rights reserved.
// *****
#include "params.h"
#include "string.h"
#include "strutils.h"

// *****
// CONSTRUCTOR FOR SIGNER PARAMS OBJECT WHICH
// MANAGES THE COMMAND LINE AND MESSAGE
// *****
SignerParams::SignerParams(LPTSTR cmd_line)
{
    char *dash_ptr; //cmd type, e.g., -commands,
    const char *dbg_msg_ptr;

    parameters.input_filename = NULL;
    parameters.output_filename = NULL;
    parameters.debug_flag = 0;
    parameters.gain = (float) 100.0;
    parameters.gamma = (float) 0.07;
    parameters.bump_size = 1;
    parameters.bit_scale = (float) 100.0;
    parameters.super_reader_flag = FALSE;
    dbg_msg_ptr = (const char *) GetMessage(1);
    TRACE(DBGING IN SignerParams constructor. Message is: %s)\n", dbg_msg_ptr);
    commands = new char[strlen(cmd_line) + 1];
    strcpy(commands, cmd_line);
    dash_ptr = NULL;
    if the command line doesn't start w/ a '-', then the command line is
    a single argument: the filename. This case comes up when the program
    is invoked by dragging a filename onto the executable in Win95 explorer.
    if (strlen(cmd_line) < 2 || !isalpha(cmd_line[0])) {
        strcpy(parameters.input_filename, cmd_line);
        strcpy(parameters.output_filename, cmd_line);
    }
    Otherwise we check for the multiple argument format of the command line,
    in which arguments pairs are used, e.g., "-f filename".
    else
    {
        do
        {
            Find the last '-' character
            dash_ptr = strrchr(cmd_line, '-');
            if (dash_ptr != NULL)
            {
                cmd_type = dash_ptr + 1;
                cmd = cmd_type + 1;
                // Create an in-core input stream
                istrstream istrstream(cmd, strlen(cmd));
                switch (cmd_type)
                {
                    case 'g':
                        istrstream >> parameters.gain;
                        break;
                    case 'gamma':
                        istrstream >> parameters.gamma;
                        break;
                    case 'b':
                        istrstream >> parameters.bump_size;
                        break;
                    case 'f':
                        istrstream >> parameters.input_filename;
                        break;
                    case 'o':
                        istrstream >> parameters.output_filename;
                        break;
                    case 'd':
                        istrstream >> parameters.debug_flag;
                        break;
                    case 'r':
                        istrstream >> parameters.super_reader_flag;
                        break;
                    default:
                        break;
                }
            }
            else
            {
                break;
            }
        } while (dash_ptr != NULL);
    }
}

//if (parameters.message == NULL)
//{
//    parameters.message = new char(strlen("Default message") + 1);
//    strcpy(parameters.message, "Default message");
//}
// Clean up
// delete [] commands;
//
// SignerParams::~SignerParams(void)
// {
//     if (parameters.input_filename != NULL)
//         delete [] parameters.input_filename;
//     //if (parameters.output_filename != NULL)
//         //delete [] parameters.output_filename;
//     //if (parameters.message != NULL)
//         //delete [] parameters.message;
//     if (parameters.output_filename != NULL)
//         delete [] parameters.output_filename;
//     if (parameters.registry_name != NULL)
//         delete [] parameters.registry_name;
// }
//
// SignerParams::UpdateSignature()
// {
//     Update the timestamp member variable within this object
//     void SignerParams::UpdateSignature(void)
//     {
//         // Set the timestamp indicating when we signed this puppy.
//         CTime t = CTime::GetCurrentTime();
//         parameters.sign_time = t;
//     }
// }
//
// FILE: Params.h
//
// DISCUSSION:
// The Params classes are responsible for gathering and managing all
// signpost parameters. There are two classes defined here: 1) the
// SignerParams class for the signer, and the ReaderParams class for the
// reader.
// The SignerParams class also keeps track of internal parameters which
// control or "tune" the operation of the signer, but which are not
// accessible by the user.
// At present, this is a non-GUI version. All
// parameters are set at compile time. In the future, a GUI version
// will be added which will present a dialog box to the user and gather
// input parameters from a graphical interface. The command line version
// will be able to handle the same parameters as the GUI version, plus
// processing. Different constructors will be used to differentiate
// between the GUI and cmd line versions.
// This header file should be included by any module which creates or
// makes use of SignerParams and/or ReaderParams objects.*

```



```

* and will make use of the public domain software LINTIFF in order*
* to read and write TIFF files.
*
* This header file should be included by any module which creates or*
* modifies the rawimage object.
*
* CREATING DATE: August 15, 1995
*
* Copyright (c) 1995 Digimarc Incorporated. All rights reserved.*
*
*#ifndef RAWIMAGE_H
*#define RAWIMAGE_H
*#include "digimarc.h"
*#include "params.h"
*
*// Since the exact internal representation may change, use a typedef.
*// This will allow a single change to modify all references to the
*// raw image data format. (Future we will need several raw image representation
*// typedefs long * Raw_Data;
*
*class RawImage
*{
*public:
*    RawImage(SignerParams *params);
*
*    // Member function which gives caller access to the raw image and its attributes.
*    const int getxdim(void);
*    const int getydim(void);
*
*    // This accessor returns a const pointer to a read-only image.
*    const Raw_Data get_image(void) const;
*
*    // This accessor returns a const pointer to a writable image.
*    Raw_Data * get_writable_image(void) const;
*
*    // Member function used to convert the raw image to an output TIFF file.
*    void tiff(char *filename);
*
*    // Private data. Users of RawImage objects get at these through accessors only.
*private:
*    int xdim; // X dimension of image
*    int ydim; // Y dimension of image
*    Raw_Data image; // Ptr to array of image data
*};
*
*#endif // RAWIMAGE_H
*
*=====
*// Main RawImage
*// DESCRIPTION: Implementation functions of the Digimarc technology
*// Created: August 1995
*// This particular code uses "raster" based processing as opposed to 2D based
*// Copyright (c) 1995 Digimarc Corporation. All rights reserved.
*//
*// Include "read.h"
*// Include "write.h"
*// Include "tiff.h"
*// Include "math.h"
*//
*// Constants
*//
*const float epsilon = (float) 0.000001;
*//
*// read_bbit_single_channel_or_color()
*//
*// Used to read or "recognize" the embedded digimarc signature in
*// either a gray-scale or color image. Set number_channels to 1 for
*// read_bbit_single_channel_or_color
*// issue data to be recognised
*//
*// It's x dimension
*//

```

[illegible]

```

        bit = key % (256 - message_length);
        bit_total[bit] += ((update_float++) - running_average) * (play_value++);
    }
    // Compute the crude metric, an estimate of rms spread of the
    // bit level detector's results. The reference bitarray is either
    // the known message (if it was available to caller) or the
    // newly computed estimate of the message.
    metric = get_crude_metric(referenceBitArray, bit_total, range, message_length);

    delete [] data_float;
    delete [] bit_total;
    delete [] bit_mag;
    delete [] bit_mag2;
    //delete [] bit_mag;

    return;
}

// =====
// flow_init
// =====
void flow_init(long x_extent, int number_channels)
{
    unsigned char *pdata;
    long i;
    float *pdata;

    pdata = data;
    if (number_channels == 1) {
        for (i = 0; i < x_extent; i++)
            *pdata++ = (float) (pdata++);
    }
    else if (number_channels == 3) {
        for (i = 0; i < x_extent; i++) {
            *pdata = (float) *pdata++;
            *pdata++ = (float) *pdata++;
            *pdata++ = (float) *pdata++;
        }
    }
}

// =====
// remove_mean
// =====
void remove_mean(float *array, long length)
{
    long i;
    float total = (float) 0.0;
    for (i = 0; i < length; i++)
        total += array[i];
    total /= (float) length;
    for (i = 0; i < length; i++)
        array[i] -= total;
}

// =====
// get_crude_metric
// =====
float get_crude_metric(
    const unsigned char *actual_message, // the original message, if you have it,
                                        // otherwise use found message
    float *bit_total,
    float *bit_mag,
    int message_length)
{
    int i;
    float avg = (float) 0.0, rms = (float) 0.0, ftemp;
    *range = (float) 0.0;

    // add up all the bits to find an average, as well as 0's
    for (i = 0; i < message_length; i++) {
        if (actual_message[i] > 0)
            avg = bit_total[i];
        else
            avg = bit_total[i];
    }
    avg /= message_length;

    // For a zero energy image, avg will equal zero. We replace it
    // with a small value.
    if (avg == 0.0)
        avg = epsilon;

    for (i = 0; i < message_length; i++)
        bit_total[i] /= avg;

    // now calculate the deviation about the nominal averages
    for (i = 0; i < message_length; i++) {
        if (actual_message[i] > 0)
            ftemp = bit_total[i] - (float) 1.0;
        else
            ftemp = bit_total[i] + (float) 1.0;
        if (fabs( (double) ftemp ) > (double) *range)
            *range = (float) fabs( (double) ftemp );
    }
    rms = ftemp * ftemp;
    ftemp = rms / ((float) message_length - (float) 1.0);
    rms = (float) sqrt(ftemp);
    return( rms ); // returns crude spread metric */
}

// =====
// int derivative_threshold(float *data, long length, int number_channels, double maxdiff, float
// filter_coef)
// =====
int derivative_threshold(float *data, long length, int number_channels, double maxdiff, float
filter_coef)
{
    long i;
    int status = 1;
    float *pdata, llast, last;
    double diff;

    float replacement = (float) 0.0;
    if (number_channels == 3) maxdiff *= 3.0;
    last = llast = data[0];
    pdata = &data[1];
}

```

[illegible]

[illegible]


```
// ReadMsg message handlers
// ReadMsg : constructor
void ReadMsg::OnOK()
{
    CDialog::OnOK();
}

// ReadMsg.h : header file
// ReadMsg : public CDialog
// Construction
ReadMsg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
//{{AFX_DATA(ReadMsg)
enum { IDD = IDD_READ_DIALOG };
int m_msg_length;
int m_msg_pos;
int m_msg_size;
float m_decali_lut_scale;
//}}AFX_DATA

// Implementation
virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support

//{{AFX_MSG(ReadMsg)
// Generated message map functions
afx_msg void OnCancel();
afx_msg void OnOK();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
```

[illegible]

```

scale=DETAIL_NORMFACTOR;
for(i=0;i<DETAIL_START;i++){detail_lut[i]=(float)1.0;
for(i=DETAIL_START;i<DETAIL_STOP;i++){
    detail_lut[i] = (float)1.0 + scale*((float)(i-DETAIL_START)/length);
for(i=DETAIL_STOP;i<DETAIL_TOTAL;i++){detail_lut[i]=detail_lut[DETAIL_STOP-1];
return(status);
}

// sign_bit_single_channel_or_color()
// written for the march 1996 bump incarnation
// unsigned char *data, // input data to be signed
// long data_length, // it's length
// long ydim, // it's y dimension
// int message, // message or message string
// unsigned char *key, // 8 bit random key, uniformly distributed
// key_length, // key_length often equal to data_length but not always
// unsigned char *key_lut, // look up table mapping key value
// float *detail_lut, // look up table mapping the scaling to luminance values
// int signing_mode, // current options: STANDARD or STRICT_LUMINANCE
// int number_channels, // added in late february 1996 to begin work on 3 color 24 bit
// int bumps, // added in March 1996 to implement bumps
){
    unsigned char *pdata;
    unsigned char *pout;
    unsigned char *pkey;
    long i;
    int j;
    int lum_change,status;
    float ftemp,delta;
    float *detail_vector, *new_float(detail);
    int key_xlength;
    key_xlength = 1*(xdim-1)/bumps;
    if(number_channels == 1){
        pdata = data;
        pkey = key;
        for(i=0;i<key_xlength;i++){
            // load local detail values for this row
            detail_vector = detail_vector + key_xlength;
            pkey+=key_xlength;
            for(j=0;j<xdim;j++){
                if(lum_change == key_bit((int)*pkey))
                    (p_out++) = *(pdata++);
                else
                    detail_vector++;
            }
        }
        else {
            local_gain = *(detail_vector); // luminance lut(detail);
            if (abs(lum_change > 1)) // this is the anti-alias check
                local_gain = (float)255.0;
            else lum_change = 1;
        }
        delta = (float)lum_change * local_gain;
        if (!(*message))
            delta = -delta; // invert current snow image luminance value ... key
        ftemp = (float)(pdata++) * delta;
        if(ftemp > (float)255.0){p_out++ = (unsigned char)255;
        else if(ftemp < (float)0.0){p_out++ = (unsigned char)0;
        else {p_out++ = (unsigned char)(ftemp/(float)0.5);
        }
        }
        if ((j+1)%bumps == 0) {

```

```

key++;
} else if ((i % bumps) * key_xlength / jumps) % message_length == (message_length - 1)) {
    // time to restart message
    message = message;
} else message++;
}

}

else if (number_channels == 3) {
    // data length is assumed to be the number of pixels, not the number of data bytes
    // so we have to figure out what order, 3 bytes in a row per pixel, A G B
    if (signing_mode == STANDARD) {
        point = data;
        for (i=0; i
```

[illegible]

```

// Set pointer to the DIB of the image which is to be saved.
m_pParams->mypp->SetParams(m_pParams->mypp->GetParams());
//TRAC ( "Params is %d", m_pParams->GetParams());
DeleteContents();
BeginWaitCursor();
// replace calls to Serialize with ReadDIBFile function
TRY
{
    m_DibOriginalDIB = ::ReadDIBFile(mFile);
}
CATCH (CException, eLoad)
{
    file.Abort(); // will not throw an exception
    EndWaitCursor();
    ReportSaveLoadException(m_pParams->GetParams(), eLoad,
        m_DibOriginalDIB == NULL,
        m_DibOriginalDIB == NULL);
    return FALSE;
}
END_CATCH

// InitDIBData();
// In debug case, dump out some information about the image.
// DumpBitmapInfoHeader();
EndWaitCursor();
if (m_DibOriginalDIB == NULL)
{
    // may not be DIB format
    MessageBox(NULL, "Couldn't load the 'Original Image'", NULL,
        MB_ICONINFORMATION | MB_OK);
    return FALSE;
}
// Save the total size needed for the DIB.
m_nDefaultDIBSize = file.GetLength() - sizeof(BITMAPFILEHEADER);

SetCaption(m_pParams->GetParams()); // start off with unmodified
SetModifiedFlag(FALSE);
// If we read an 8 or 24 bit image, we're finer, else warn user
if (m_BitsPerPixel == 8 || m_BitsPerPixel == 24)
{
    else
    {
        MessageBox(NULL, "The file doesn't contain an 8 or 24 bit image.\n"
            "It will be displayed, but can't be signed or read.",
            "Original Signat Warning", MB_ICONINFORMATION | MB_OK);
    }
    return TRUE;
}
// OnSaveDocument()
// m_nDocDoc: OnSaveDocument(const char* pCaptionName)
{
    file:
    CFile
    CFileException
    view_Type:
    DIB
    if (!file.Open(m_pParams->GetParams(), CFile::modeCreate |
        CFile::modeReadWrite | CFile::shareExclusive, 4096))
    {
        ReportSaveLoadException(m_pParams->GetParams(), eFile,
            return FALSE;
    }
}
// replace calls to Serialize with savedDIB function
B000_Success = FALSE;
// Determine which DIB to save, based on the active window.
view_Type = GetActiveViewType();

```



```

float *luminance_lut = new float[256];
//::GlobalUnpack Lut(luminance_lut, m_params->dstCnms());
}

// Create and load the key look up table.
char *key_lut = new char[256];
m_params->key_lut_lut_lut, m_params->getGain();

long data_length = unsignedImage.GetXDim() * unsignedImage.GetYDim();
// Create a packed msg (will be a user input in future).
// (this is a user input in future)
m_packedMsg = new PackedMsg( (const char *) m_params->dstMessage());

// Set up some arguments and call the core signer.
int x_dim = unsignedImage.GetXDim();
int y_dim = unsignedImage.GetYDim();
int num_channels = 1;
if (unsignedImage.GetBitsPerPixel() == 24)
else if (unsignedImage.GetBitsPerPixel() == 8)
{
// core_sign_lut(0, (float) 0.0); // Later this will be user controlled
float *detail_lut = new float[256];
load_detail_lut(detail_lut, m_params->detailScale());
// sign_rgb_single_channel_or_color(unsignedImage.GetPackedData(),
data_length,
y_dim,
x_dim,
m_packedMsg->getSignKeyArray(),
snowyImage.GetPackedData(),
data_length,
luminance_lut,
key_lut,
STRAND);
signedImage.GetPackedData(),
m_params->dstBumpSize());
}

delete [] detail_lut;
// Set the timestamp indicating when we signed this puppy.
m_params->updateSignature();

delete [] luminance_lut;
delete [] key_lut;
// Now unpack the data in the image object, back into the standard DIB format
signedImage.UnpackData();
}

// Read()
// The read function is the interface to the core recognition algorithm
// and takes the necessary data structures needed by the core routine
// and makes the call.
// Read Color: Read DIB's signedData, now use super_reader
{
long num_pixels, num_colors;
int reading mode;
// Create Image objects for the images. Note that this loads them in memory.
Image snowyImage(m_snowyDIB);
Image signedImage(m_signedDIB);
// Create a "byte-wise" packed data array from the DIB 4-byte packing
signedImage.MakePackedData();
snowyImage.MakePackedData(FORCE_TO_1_CHANNEL); // Snowy images always 1 ch.
// unsignedImage.MakePackedData();

num_pixels = (long) signedImage.GetXDim() * signedImage.GetYDim();
num_colors = signedImage.GetNumColors();

if (m_bitsPerPixel == 8 && m_bitsPerPixel != 24)
}

TRACE("At this time, only build snow image for 8 or 24 bit images\n");
//::GlobalUnpack((HGLOBAL) m_snowyDIB);
return;
}

if (m_bitsPerPixel == 8 || m_bitsPerPixel == 24)
{
// Create key for (m_params->dstKey(), (BITMAPINFO *) &snowyDIB,
// snowyDIB);
}

//::GlobalUnpack((HGLOBAL) m_snowyDIB);
}

// Sign()
// This is the function which calls upon the core signing algorithms.
// MAKING COMPLEXITY THIS FUNCTION ASSUMES THAT WE ALWAYS ARE SIGNING
// THE ORIGINAL IMAGE DIB. THIS MAY NOT BE A BUG.
// The function which calls the signer core algorithm
void CImage::Sign(void)
{
long num_pixels, num_colors;
int image_type;
src_data, dest_data; // Huge ptrs for copying the image.
int x_dim, y_dim;
int num_channels;
// IDB horizontalDIB = GetOriginalDIB();
// if (m_bitsPerPixel == 8)
// {
// MessageBox(NULL,
// "Insufficient memory is available for the signed image",
// "Insane Signer Warning",
// MB_OK);
// }
return;

// Create image objects for the images. Note that this loads them in memory.
// Create signedImage(m_signedDIB);
// This is only, but I have to copy the DIB header stuff into the signed DIB
// before I can create the signedImage object.
dest_data = (char *) GlobalLock((HGLOBAL) m_signedDIB);

// We want to copy the BITMAPINFO structure from the unsigned to the signed DIB
src_data = GlobalLock((HGLOBAL) m_snowyDIB);

// Copy the BITMAPINFOHEADER and palette to the signed DIB space, byte by byte.
for (image_type = 0; image_type < unsignedImage.GetSizeOfHeader(); image_type++)
{
*dest_data++ = *src_data++;
}

//::GlobalUnpack((HGLOBAL) m_signedDIB);
// Now create the signedImage object, which will lock the DIB in memory again.
Image signedImage(m_signedDIB);

// For each, create a "byte-wise" packed data array from the DIB 4-byte packing
snowyImage.MakePackedData(FORCE_TO_1_CHANNEL); // Snowy image always 1 chan
signedImage.MakePackedData();

num_pixels = (long) unsignedImage.GetXDim() * unsignedImage.GetYDim();
num_colors = unsignedImage.GetNumColors();

if (m_bitsPerPixel == 8 && m_bitsPerPixel != 24)
{
TRACE("At this time, only sign 8 and 24 bit images\n");
return;
}

// Create and load the luminance scaling look up table.

```

[illegible]

```

{
    // Run the reader again to see if we recover message.
    Read(m_signals, FALSE);

    m_state = IMAGE_SIGNED_AND_VERIFIED;

    // Now see if a "signed image" view exists. If not, create it.
    CreateUniqueView(SIGNED_VIEW);

    // Now see if a "status image" view exists. If not, create it.
    CreateView "p_statusview"
    p_statusview = (CdbView *) CreateUniqueView(STATUS_VIEW);

    RedrawCursor();

    // Refresh all of the views (Don't actually need to refresh Original one)
    p_statusview->Refresh();

    UpdateAllViews(NULL);

    // Some debug stuff related to checksums.
    TRACE("Before computed checksum: %X\n", (int) m_PackedImage->GetHeaderChecksum());
    TRACE("Reader computed checksum: %X\n", (int) m_PackedImage->GetHeaderChecksum());
}

}

CreateUniqueView()
// This function creates a new view of the indicated type, if and
// only if one does not already exist, or returns a pointer to
// the view if one does already exist. It also returns a pointer to
// pre-existing view of the specified type if one already exists.
// The view type is indicated by the argument.
// i.e. SIGNED_VIEW, ORIGINAL_VIEW, STATUS_VIEW types from SigntView.h.
// Caller must ensure that the view is not NULL.
// Caller: CDBOBC: CreateUniqueView(int view_type)
//
// BOOL view found = FALSE;
// CView* pview;
// while (pos != NULL)
// {
//     pview = GetNextView(pos);
//     // If we find it, we return the pointer and we're done.
//     if ( ((CdbView*)pview)->GetViewType() == view_type )
//         return pview;
// }
//
// The desired type of view doesn't exist, so we create it.
//
// CreateFrame mainFrame = (CMainFrame *) AfxGetApp()->m_mainFrame;
// mainFrame->AddChildWindow(pview);
//
// Now find the empty created view (last in list) and set its type.
// pos = GetFirstViewPosition();
// while (pview = GetNextView(pos);
// {
//     ((CdbView*)pview)->SetViewType(view_type);
// }
// return(pview);
}

// ChangeViewType()
//
// This function finds the view of the "old type", and changes its
// type to "new type". If successful, it returns a pointer to
// the view. If not successful, it returns NULL.
// The "view type" arguments are from the view types in SigntView.h.
// i.e. SIGNED_VIEW, ORIGINAL_VIEW, STATUS_VIEW, ALIGNED_VIEW.
// Caller must ensure that the view is not NULL.
// Caller: CDBOBC: ChangeViewType(int old_type, int new_type)
//
// BOOL view found = FALSE;
// POSITION pos = GetFirstViewPosition();
// while (pos != NULL)
// {
//     pview = GetNextView(pos);
//     // If we find it, change its type to return the pointer and we're done.
//     if ( ((CdbView*)pview)->GetViewType() == old_type )
//     {

```

[illegible]

```
// m_autoread, we set or clear the check mark next to
// the menu item using the pwnd()-setCheck() function.
// If the menu item is checked, we call the m_align()
void CFileDialog::OnPackedSettingAutoRead(CCmdUI *pCmdUI)
{
    // Set or clear the check mark in the menu
    if ((m_bBookmarks + !GetApp()) -> m_autoread == TRUE)
        pCmdUI->setCheck(FALSE);
    else
        pCmdUI->setCheck(TRUE);
}

// =====
// OnSetAlignDialog()
// This function is called when the user selects the "Align" menu option.
// A CFileDialog object is created and used in order for the operator
// to specify the name of the "reference image" (a signed or unsigned
// integer value). The reference image is used to align the suspect
// image.
void CFileDialog::OnSetAlignDialog()
{
    CFileDialog *pDlg;
    BOOL success_flag;

    // Create a filter for the types of files the file dialog will offer
    char szFilter[100] = "Image Files (*.bmp;*.dib)|*.bmp;*.dib|*.*";
    // All files (*.*) *.*
    pDlg = new CFileDialog(TRUE, "", szFilter, OFN_HIDEREADONLY | OFN_WRITEITEMSNOTSORTED);
    // It's a file open (not save) dialog
    if (!pDlg->DoModal()) return;
    // Construct a file dialog
    CFileDialog *pFileDlg = new CFileDialog(TRUE, "", "", OFN_HIDEREADONLY | OFN_WRITEITEMSNOTSORTED);
    // Display the file dialog
    if (!pFileDlg->DoModal()) return;
    // Get the name of the reference image file.
    CString refname = pFileDlg->GetPathName();
    BeginWaitCursor();
    // Create an Image object for the reference image.
    // If one already exists, delete it first.
    if (m_pRefImage != NULL) delete m_pRefImage;
    m_pRefImage = new Image(refname);
    // bail out if something went wrong
    if (!m_pRefImage->IsValid() || m_pRefImage->GetFormat() != IMAGE_BITMAP) return;
    // Display the reference image
    CreateOnScreenView(m_pRefImage, VIEW);
    UpdateAllViews(NULL);
}

// =====
// TRACE("Call the Align() function (this is a test of trace output.)\n");
// Do the actual alignment and change update the state description.
success_flag = AlignIt();
if (success_flag)
{
    m_status = SUSPECT_ALIGNED;
    // Now, the template image object has had its packed data array replaced
    // by the aligned, co-extensive image. Need to move this packed data
    // from the original image to the aligned image (and possible file saving) purposes.
    void CImage::InheritData(CImage *) const
    {
        // We now call the image the Aligned Image, not reference
        m_AlignedImage = m_pRefImage;
        m_pRefImage = NULL;
        CreateOnScreenView(ALIGNED_VIEW);
        // Create a status view, if it doesn't already exist.
        CStatusView *pStatusView;
        p_StatusView = (CStatusView *) CreateOnScreenView(STATUS_VIEW);
        p_StatusView->Init();
        UpdateAllViews(NULL);
    }
}
```

[illegible]

[illegible]

[illegible][illegible]

[illegible]

```

# END BASE LINK32 addresses lib /nologo /check:0x2600 /subsystem:windows /debug /machine:IA64
LINK32 /add-ordinal:ms lib /nologo /check:0x2600 /subsystem:windows /profile
"DBP_PFILE" /out:"%OUTDIR%\SignerWin32.exe"
"%SIGNED%"
%*%SIGNED%
#####
*$(INTDIR)\Vbapi.obj" : $(SOURCE) $(DBP_CFP_DTBAP) *$(INTDIR)"
# End Source File
# Begin Source File
SOURCE= \Image.cpp
DBP_CFP_IMAGES= \
    *.Image.h^
    *.Vbapi.h^
    *.Vcdata.h^
*$(INTDIR)\Image.obj" : $(SOURCE) $(DBP_CFP_IMAGES) *$(INTDIR)"
*$(INTDIR)\Image.abr" : $(SOURCE) $(DBP_CFP_IMAGES) *$(INTDIR)"
#####
# End Source File
# Begin Source File
SOURCE= \Mainfrm.cpp
DBP_CFP_MAINFRM= \
    *.Mainfrm.h^
    *.Vcdata.h^
*$(INTDIR)\Mainfrm.obj" : $(SOURCE) $(DBP_CFP_MAINFRM) *$(INTDIR)"
*$(INTDIR)\Mainfrm.abr" : $(SOURCE) $(DBP_CFP_MAINFRM) *$(INTDIR)"
#####
!ELSEIP *$(CFG)" == "Signer - Win32 Debug"
DBP_CFP_MAINFRM= \
    *.Mainfrm.h^
    *.Vcdata.h^
*$(INTDIR)\Mainfrm.obj" : $(SOURCE) $(DBP_CFP_MAINFRM) *$(INTDIR)"
*$(INTDIR)\Mainfrm.abr" : $(SOURCE) $(DBP_CFP_MAINFRM) *$(INTDIR)"
#####
!ENDIF
#####
# End Source File
# Begin Source File
SOURCE= \VbFile.cpp
DBP_CFP_VBFILE= \
    *.Vbdata.h^
    *.Vcdata.h^
*$(INTDIR)\VbFile.obj" : $(SOURCE) $(DBP_CFP_VBFILE) *$(INTDIR)"
*$(INTDIR)\VbFile.abr" : $(SOURCE) $(DBP_CFP_VBFILE) *$(INTDIR)"
#####
# End Source File
# Begin Source File
SOURCE= \Packmsg.cpp
DBP_CFP_PACKMSG= \
    *.Packmsg.h^
    *.Vcdata.h^
*$(INTDIR)\Packmsg.obj" : $(SOURCE) $(DBP_CFP_PACKMSG) *$(INTDIR)"
*$(INTDIR)\Packmsg.abr" : $(SOURCE) $(DBP_CFP_PACKMSG) *$(INTDIR)"
#####
# End Source File
# Begin Source File
SOURCE= \Vbapi.cpp
DBP_CFP_VBAPI= \
    *.Vbapi.h^
*$(INTDIR)\Vbapi.obj" : $(SOURCE) $(DBP_CFP_VBAPI) *$(INTDIR)"

```

[illegible]

[illegible]


```

{
    TRACE("SelectPalette failed in CDView::OnPaletteChanged()");
}

return m_h;
}

// OnInitialUpdate()
void CDView::OnInitialUpdate()
{
    ASSERT(GetDocument() != NULL);

    SetScrollSizes(MM_TEXT, GetDocSize());

    // Resize this view's window based on the size of the image.
    ResizeParentToFit();

    GetParent()->SetWindowText(GetDocument()->GetTitle() + " - Original");

    OnActivateView()

    void CDView::OnActivated(BOOK_Activate, CView_ActivateView,
        CView_ActivateView)

    {
        ScrollBar::OnActivate(BarDeactivate, DeactivateView, DeactivateView);

        if (BarDeactivate)
            m_hBitmapActive = TRUE;
        ASSTRT(GetActiveView() == this);
        OnInitialize(WINDOW_MIM_0); // same as SendMessage(WM_DORESTORE);
    }
    else
        m_hBitmapActive = FALSE;

    OnPaintCopy()

    void CDView::OnPaintCopy()
    {
        CDibObj* pDib = GetDocument();
        // Clean clipboard of contents, and copy the DIB.
        if (OpenClipboard())
        {
            RegDeleteCursor();
            RegDeleteClipboardData(ClipboardData);
            CloseClipboard();
            RedrawCursor();
        }

        OnPaintEditCopy()

        void CDView::OnPaintEditCopy(CCmdT* pCmdT)
        {
            CDibObj* pDib = GetDocument();
            // Set the window title
            GetParent()->SetWindowText(GetDocument()->GetTitle() + " - Code Pattern");

            pCmdT->Enable(GetIDB() != NULL);
        }

        OnPaintPanel()

        void CDView::OnPaintPanel()
        {
            IDB hNewIDB = NULL;
            if (OpenClipboard())
            {
                RegDeleteCursor();

                hNewIDB = (IDB) CopyHandle((COPY_BoardData (CP_DIB));
                CloseClipboard();

                if (hNewIDB != NULL)

```


[illegible]

[illegible]

[illegible]